**CHAPTER 1**

# INTRODUCTION

Delta Modulation (DM) is a simple form of audio compression where each sample is encoded as a difference (delta) from the previous sample, rather than as an absolute value. This process can result in significant data compression, especially when the signal has gradual changes.Audio compression using Delta Modulation in Python involves processing each sample as a delta (difference) from the previous one, then encoding this delta as a single bit. Audio compression is essential for efficiently storing, transmitting, and processing audio data in today’s digital world. As audio files can be large and bandwidth can be limited, compression methods are used to reduce file sizes while maintaining acceptable audio quality.

There are two main types of audio compression: lossy and lossless. Lossy compression methods, such as MP3 and AAC, reduce file sizes significantly by discarding some audio data deemed less critical, making them popular for music streaming and downloads. Lossless compression methods like FLAC and ALAC, however, preserve the entire audio signal, making them preferred by audiophiles and professionals who need high fidelity. Additionally, techniques like Delta Modulation (DM) and Adaptive Differential Pulse Code Modulation (ADPCM) offer lightweight, low-power solutions suitable for real-time applications and embedded systems.

Delta Modulation, a simpler form of audio compression, encodes audio as a series of small changes (deltas) from one sample to the next rather than absolute values. This low-complexity method is particularly useful for voice data and embedded systems where efficiency is critical. By exploring Delta Modulation, we can better understand its potential as a viable compression technique for applications where computational simplicity, energy efficiency, and real-time performance are necessary.

**CHAPTER 2**

**PROBLEM STATEMENT**

* Audio data often requires significant storage and bandwidth.Compression techniques, like Delta Modulation (DM) and Differential Pulse Code Modulation (DPCM), can efficiently reduce file size.
* Delta modulation uses a simpler encoding and decoding process compared to DPCM. Delta Modulation encodes the signal by just tracking changes (whether the signal goes up or down) instead of encoding the full difference from the previous sample.
* Delta modulation only transmits one bit per sample, which can help in reducing bandwidth requirements. In contrast, DPCM often requires more bits to represent the sample difference, especially if the signal changes rapidly.
* Delta Modulation generally requires simpler circuitry for both encoding and decoding, making it less expensive and easier to implement in hardware. DPCM requires more complex hardware due to its more intricate difference calculations.
* Delta modulation is more robust to certain types of noise since only the change in the signal is transmitted. Errors in Delta Modulation tend to have less noticeable impact than in DPCM, which can make it advantageous in noisy environments.

**CHAPTER 3**

**MARKET SURVEY**

**Target Sectors for Delta Modulation in Audio Compression**

• Companies like Fraunhofer (AAC), Dolby (AC-4), and Qualcomm (AptX) dominate advanced audio compression markets, primarily focusing on high-fidelity audio for music, streaming, and media.

• Telecommunications companies and manufacturers of embedded systems often utilize simpler, low-complexity compression methods like Delta Modulation, especially for voice and low-bitrate audio streaming (e.g., walkie-talkies, simple IoT audio devices).

• Delta Modulation can be a cost-effective solution in voice communication systems like two-way radios, walkie-talkies, and VoIP systems where low data rates and low power consumption are essential.

• IoT applications, such as smart speakers and wearable devices, often require real-time voice transmission. DM’s low complexity makes it appealing in these applications where hardware resources are constrained.

**Current Audio Compression Techniques**

1. **Lossy Compression**

• **MP3 (MPEG Audio Layer III)**: One of the most popular audio formats, MP3 achieves high compression by discarding audio data considered inaudible to human ears, balancing quality and file size.

• **AAC (Advanced Audio Coding)**: Known for better quality at similar bitrates compared to MP3, AAC is widely used in streaming applications and by platforms like YouTube and Apple Music.

• **Opus**: A versatile codec optimized for real-time applications, Opus adapts to different bitrates and is widely used in VoIP, streaming, and video conferencing.

2. **Lossless Compression**

• **FLAC (Free Lossless Audio Codec)**: FLAC reduces file size without losing any audio data, ideal for music enthusiasts who prefer high-quality sound. FLAC files are compressed but fully reversible to the original quality.

• **ALAC (Apple Lossless Audio Codec)**: Similar to FLAC but designed for Apple devices, ALAC retains the original audio quality while reducing file size.

**CHAPTER 4**

**MATHEMATICAL MODEL**

Delta Modulation (DM) compresses an audio signal by encoding each sample as the difference (delta) from the previous sample. This technique relies on quantizing the difference between the current sample and a predicted sample value, rather than storing the absolute value of each sample.

1. **Signal Representation**:

Let an audio signal be represented as a sequence of samples  , where  n  represents discrete time.

2. **Prediction Step**:

In DM, each sample  x[n]  is approximated by a predicted sample  x[n] . For simplicity, DM uses the previous reconstructed sample as the prediction, so:

where  y[n-1]  is the reconstructed sample at time  n-1 .

3. **Difference Calculation (Delta)**:

The delta  d[n]  is calculated as the difference between the actual sample and the predicted sample:

4. **Quantization (Encoding Step)**:

In basic DM, the delta  d[n]  is quantized to a single bit:

• If  , encode  d[n]  as 1 (indicating an increase).

• If  , encode  d[n]  as 0 (indicating a decrease).

5. **Step Size Adjustment**:

A fixed step size    is used to update the reconstructed signal. The reconstructed sample  y[n]  is generated as:

y[n] = x[n] + s[n].

where  s[n]  is the encoded bit:

• s[n] = +1  if

• s[n] = -1  if  d[n] < 0

6. **Reconstruction (Decoding)**:

The output (reconstructed signal)  y[n]  is obtained iteratively as:

This iterative addition or subtraction of the step size based on the encoded bit regenerates an approximation of the original signal.

**CHAPTER 5**

**METHODOLOGY**

**Data Collection**

• **Audio Sample Selection**: Select a .wav audio file, ideally with gradual signal changes, such as voice data.

• **Preprocessing**: Load and normalize the audio signal for consistent processing.

**Delta Modulation (Compression)**

• **Calculate Deltas**: For each sample, compute the difference from the previous sample.

• **Binary Encoding**: Encode each difference as a binary value (1 for positive change, 0 for negative change), storing only changes, not absolute values.

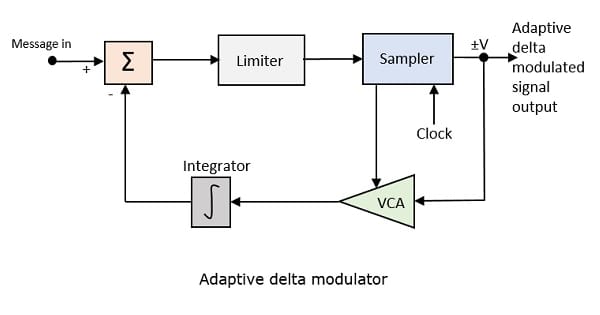
**Delta Demodulation (Decompression)**

• **Decode Signal**: Using the binary values, reconstruct the audio signal by cumulatively adding or subtracting a fixed step size based on the encoded bit (1 or 0).

**Evaluation**

• **Comparison**: Plot and compare the original and decompressed signals to visually assess fidelity.

• **Save Output**: Save the decompressed audio as a .wav file to measure reconstruction accuracy.



The basic expression can be represented as:

Where:

= current sample value

= previous sample value

= delta modulation output (1 or 0)

**CHAPTER 6**

**CODE**

import numpy as np

import matplotlib.pyplot as plt

import soundfile as sf

def delta\_modulation(input\_signal, delta):

compressed\_signal = np.zeros(len(input\_signal))

predicted\_signal = np.zeros(len(input\_signal))

for i in range(1, len(input\_signal)):

if input\_signal[i] > predicted\_signal[i-1]:

compressed\_signal[i] = 1

predicted\_signal[i] = predicted\_signal[i-1] + delta

else:

compressed\_signal[i] = -1

predicted\_signal[i] = predicted\_signal[i-1] - delta

return compressed\_signal, predicted\_signal

def delta\_demodulation(compressed\_signal, delta):

decompressed\_signal = np.zeros(len(compressed\_signal))

for i in range(1, len(compressed\_signal)):

decompressed\_signal[i] = decompressed\_signal[i-1] + compressed\_signal[i] \* delta

return decompressed\_signal

# Load an audio file (e.g., a .wav file)

input\_signal, sample\_rate = sf.read('input.wav')

# Normalize the input signal

input\_signal = input\_signal / np.max(np.abs(input\_signal))

# Set delta (step size for delta modulation)

delta = 0.01

# Delta Modulation (Compression)

compressed\_signal, predicted\_signal = delta\_modulation(input\_signal, delta)

# Delta Demodulation (Decompression)

decompressed\_signal = delta\_demodulation(compressed\_signal, delta)

# Save the decompressed signal as an output audio file

sf.write('output\_audio.wav', decompressed\_signal, sample\_rate)

# Plot the original, compressed, and decompressed signals

plt.figure(figsize=(15,5))

plt.subplot(3,1,1)

plt.plot(input\_signal)

plt.title('Original Signal')

plt.subplot(3,1,2)

plt.plot(compressed\_signal)

plt.title('Compressed Signal (Delta Modulation)')

plt.subplot(3,1,3)

plt.plot(decompressed\_signal)

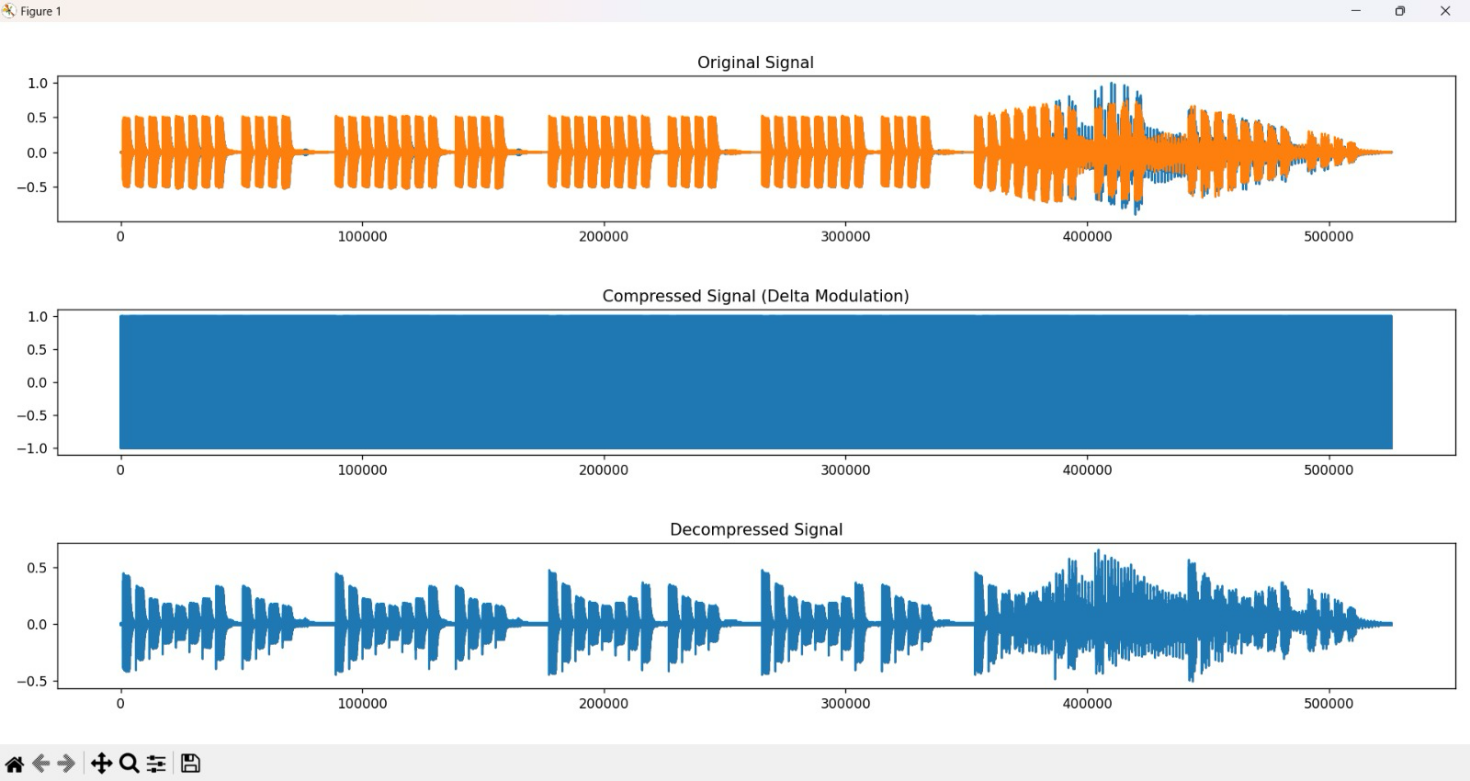
plt.title('Decompressed Signal')

plt.tight\_layout()

plt.show()

**CHAPTER 7**

**OUTPUT**



**CHAPTER 8**

**CONCLUSION**

In conclusion, Delta Modulation (DM) proves to be a simple yet effective method for audio compression, particularly for signals with gradual changes, such as voice data. Its low computational complexity makes it well-suited for embedded systems, real-time applications, and low-power devices. Delta Modulation’s efficiency in reducing data size supports storage and transmission needs in scenarios with limited resources, making it valuable for applications like telecommunications and IoT devices. However, while Delta Modulation is effective for certain audio types, more complex signals may require adaptive techniques, such as variable step sizes, to improve quality. This could broaden Delta Modulation’s applications and enhance its utility across more demanding audio compression scenarios.